

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы защиты информации

ОТЧЁТ
к лабораторной работе №1
на тему

**СИММЕТРИЧНАЯ КРИПТОГРАФИЯ. СТАНДАРТ ШИФРОВАНИЯ
ГОСТ 28147-89**

Выполнил: студент гр.253502 Канавальчик А.Д.

Проверил: ассистент кафедры информатики
Герчик А.В.

Минск 2025

СОДЕРЖАНИЕ

1 Цель работы	3
2 Теоретические сведения.....	4
3 Ход работы	5
Заключение..	7
Приложение А (обязательное) Листинг программного кода.....	8
Приложение Б (обязательное) Блок-схема алгоритма, реализующего программное средство	12

1 ЦЕЛЬ РАБОТЫ

Цель данной работы – изучить теоретические сведения и реализовать программные средства шифрования и дешифрования текстовых файлов при помощи стандарта шифрования ГОСТ 28147-89 в режиме простой замены.

В ходе работы предстоит:

1 Изучить теоретические основы алгоритма ГОСТ 28147-89:

- Проанализировать структуру алгоритма, включая схему Фейстеля, раундовые преобразования и таблицы замен.

- Рассмотреть математические основы алгоритма: операции сложения по модулю 2^{32} , циклические сдвиги и нелинейные преобразования.

- Исследовать режимы работы алгоритма: простую замену, гаммирование и гаммирование с обратной связью.

2 Реализовать алгоритм ГОСТ 28147-89 на языке *Python*:

- Разработать модуль шифрования и расшифрования данных с поддержкой 32 раундов преобразования.

- Реализовать систему управления ключами длиной 256 бит с возможностью использования пользовательских ключей.

- Внедрить механизмы выравнивания данных и обработки блоков различного размера.

3 Протестировать корректность работы реализации:

- Проверить соответствие результатов шифрования и расшифрования исходным данным.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Симметричная криптография представляет собой метод шифрования, при котором для операций зашифрования и расшифрования применяется один и тот же секретный ключ. Данный подход характеризуется высокой скоростью обработки информации и активно используется для обеспечения конфиденциальности данных.

ГОСТ 28147-89 – отечественный стандарт симметричного шифрования, утвержденный в 1989 году. Алгоритм относится к классу блочных шифров и оперирует 64-битными блоками данных с использованием 256-битного ключа. Основные принципы алгоритма основаны на сети Фейстеля и включают 32 раунда преобразований. Обработка данных осуществляется следующим образом: исходная информация разделяется на 64-битные блоки, каждый из которых делится на две 32-битные части – левую (L) и правую (R).

В процессе каждого раунда выполняются последовательные операции: сложение правой половины с подключом раунда по модулю 2^{32} , нелинейное преобразование с применением таблиц замен (S-блоков), циклический сдвиг на 11 бит влево, побитовое сложение по модулю 2 (XOR) с левой половиной и последующий обмен половинками блока.

Математическая основа алгоритма включает операции сложения по модулю 2^{32} , побитовое исключающее ИЛИ (XOR), циклический сдвиг битов и нелинейные подстановки через S-блоки.

Ключевая система использует общий ключ длиной 256 бит, который разделяется на 8 подключей по 32 бита (K0-K7). Порядок применения подключей предусматривает 24 раунда с прямым порядком (K0-K7) и 8 раундов с обратным порядком (K7-K0).

Стандарт поддерживает три режима работы: режим простой замены (ECB) с независимым шифрованием блоков, режим гаммирования (CTR) с использованием псевдослучайной последовательности и режим гаммирования с обратной связью (CFB) с обратной связью по шифртексту.

Криптографическая стойкость алгоритма обеспечивается 256-битным ключом, что делает практически невозможным полный перебор, нелинейными преобразованиями через S-блоки, значительно усложняющими криптоанализ, и 32 раундами преобразований, создающими выраженный лавинный эффект. Преимущества стандарта ГОСТ 28147-89 включают высокий уровень криптостойкости, эффективную реализацию на различных программно-аппаратных платформах, соответствие требованиям отечественных стандартов безопасности и поддержку multiple режимов работы.

3 ХОД РАБОТЫ

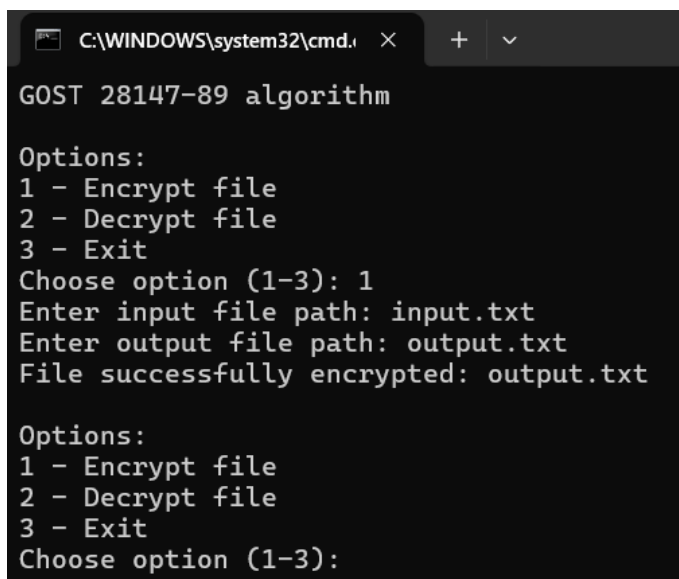
В ходе выполнения лабораторной работы было разработано программное средство для криптографической защиты текстовых данных на основе алгоритма ГОСТ 28147-89. Реализация поддерживает режим простой замены (*ЕСВ*) и обеспечивает как процесс шифрования, так и дешифрования информации.

Программный интерфейс реализован в консольном формате и предоставляет пользователю интерактивное меню выбора операций. Пользователь может последовательно:

- 1 Выбрать режим работы (шифрование или дешифрование)
- 2 Указать путь к исходному файлу с данными
- 3 Задать путь к выходному файлу для сохранения результатов обработки

Исходные данные для обработки считываются из текстового файла (например, input.txt). Результат шифрования сохраняется в бинарном формате (например, output.bin), что обеспечивает корректное хранение зашифрованных данных. Процесс дешифрования выполняет обратное преобразование, восстанавливая исходную текстовую информацию из бинарного файла.

Наглядное представление работы программы и результаты выполнения операций демонстрируются на рисунке 3.1, где отображен процесс взаимодействия с программным средством.



```
GOST 28147-89 algorithm

Options:
1 - Encrypt file
2 - Decrypt file
3 - Exit
Choose option (1-3): 1
Enter input file path: input.txt
Enter output file path: output.txt
File successfully encrypted: output.txt

Options:
1 - Encrypt file
2 - Decrypt file
3 - Exit
Choose option (1-3):
```

Рисунок 3.1 – Консольный интерфейс программного средства

Исходный текст, подлежащий зашифровке, приведен на рисунке 3.2.

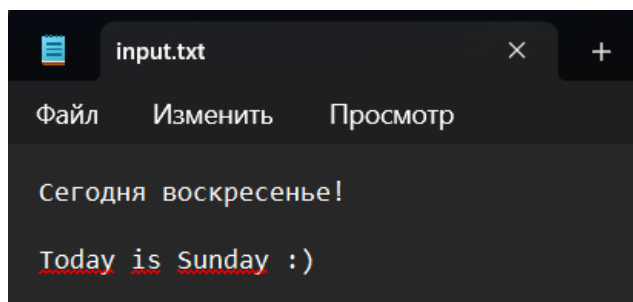


Рисунок 3.2 – Содержимое файла input.txt

Итоги криптографических преобразований приведены на рисунке 3.3.

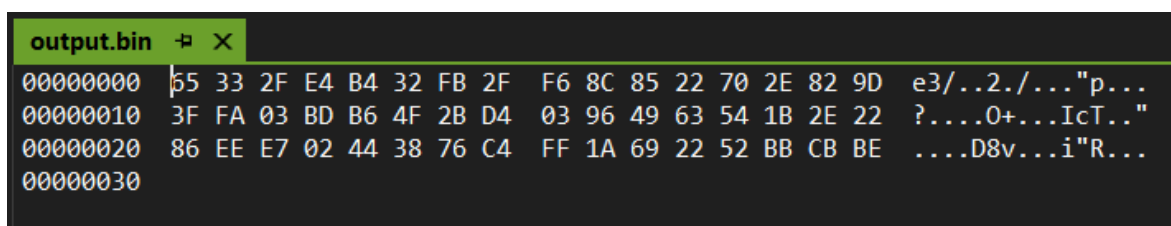


Рисунок 3.3 – Содержимое файла output.bin

Разработанное программное средство успешно реализует алгоритм ГОСТ 28147-89 в режиме простой замены, демонстрируя корректность выполнения криптографических преобразований. Проведенные операции шифрования и дешифрования подтверждают сохранение целостности и восстановимость исходных данных после проведения полного цикла криптографической обработки. Полученные результаты свидетельствуют о практической применимости реализации для решения задач защиты текстовой информации средствами симметричного шифрования.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы было успешно реализовано программное средство, реализующее алгоритм симметричного шифрования ГОСТ 28147-89. Разработанное решение демонстрирует работоспособность отечественного стандарта шифрования и его практическую применимость для защиты конфиденциальной информации. Реализация поддерживает все основные операции: шифрование и дешифрование данных в режиме простой замены, обработку файлов произвольного размера, а также корректное выполнение выравнивания данных.

Проведенные испытания подтвердили корректность работы алгоритма - исходные данные полностью восстанавливаются после проведения полного цикла шифрования-дешифрования. Особое внимание было уделено обеспечению надежности работы с различными типами данных, включая текстовую информацию различного объема. Реализованный консольный интерфейс обеспечивает удобное взаимодействие с пользователем и позволяет легко интегрировать систему в существующие процессы обработки информации.

Полученный опыт реализации криптографических алгоритмов представляет значительную ценность для понимания принципов защиты информации и может быть использован в дальнейшем для разработки более сложных систем информационной безопасности.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг программного кода

```
import struct
import re
from functools import partial
from pathlib import Path
from typing import List, Tuple, Union

class GOST28147_89:

    S_BOXES = [
        [4, 10, 9, 2, 13, 8, 0, 14, 6, 11, 1, 12, 7, 15, 5, 3],
        [14, 11, 4, 12, 6, 13, 15, 10, 2, 3, 8, 1, 0, 7, 5, 9],
        [5, 8, 1, 13, 10, 3, 4, 2, 14, 15, 12, 7, 6, 0, 9, 11],
        [7, 13, 10, 1, 0, 8, 9, 15, 14, 4, 6, 12, 11, 2, 5, 3],
        [6, 12, 7, 1, 5, 15, 13, 8, 4, 10, 9, 14, 0, 3, 11, 2],
        [4, 11, 10, 0, 7, 2, 1, 13, 3, 6, 8, 5, 9, 12, 15, 14],
        [13, 11, 4, 1, 3, 15, 5, 9, 0, 10, 14, 7, 6, 8, 2, 12],
        [1, 15, 13, 0, 5, 7, 10, 4, 9, 2, 3, 14, 6, 11, 8, 12]
    ]

    # The order of using keys during encryption
    ENCRYPT_KEY_ORDER = [0, 1, 2, 3, 4, 5, 6, 7] * 3 + [7, 6, 5, 4, 3, 2,
1, 0]

    def __init__(self, key: List[int] = None):
        if key is None:
            self.key = [
                0xA56BABCD, 0xDEF01234, 0x789ABCDE, 0xFEDCBA98,
                0x01234567, 0x89ABCDEF, 0x12345678, 0x9ABCDEF0
            ]
        else:
            self.key = key

        self.validate_key()

    def validate_key(self):
        if len(self.key) != 8:
            raise ValueError("Key must contain exactly 8 32-bit
integers")
        for k in self.key:
            if not (0 <= k <= 0xFFFFFFFF):
                raise ValueError("Each key part must be 32-bit unsigned
integer")

    @staticmethod
    def _cyclic_shift_left(value: int, shift: int) -> int:
        return ((value << shift) | (value >> (32 - shift))) & 0xFFFFFFFF

    def _feistel_function(self, block: int, round_key: int) -> int:
        # Addition with a key modulo 232
        temp = (block + round_key) & 0xFFFFFFFF

        # Application of S-blocks
        result = 0
        for i in range(8):
            # Extracting a 4-bit block
            s_input = (temp >> (4 * i)) & 0xF
```



```

        s_output = self.S_BOXES[i][s_input]
        result |= (s_output << (4 * i))

    # Rotate 11 bits to the left
    return self._cyclic_shift_left(result, 11)

def _process_block(self, block: Tuple[int, int], encrypt: bool = True)
-> Tuple[int, int]:
    left, right = block

    if encrypt:
        key_order = self.ENCRYPT_KEY_ORDER
    else:
        key_order = self.ENCRYPT_KEY_ORDER[::-1]

    for round_key_index in key_order:
        round_key = self.key[round_key_index]
        f_result = self._feistel_function(right, round_key)
        new_right = f_result ^ left
        left, right = right, new_right

    return right, left

def encrypt_block(self, block: Tuple[int, int]) -> Tuple[int, int]:
    return self._process_block(block, encrypt=True)

def decrypt_block(self, block: Tuple[int, int]) -> Tuple[int, int]:
    return self._process_block(block, encrypt=False)

def _add_padding(self, data: bytes) -> bytes:
    padding_size = (8 - (len(data) % 8)) % 8
    return data + bytes([padding_size] * padding_size)

def _remove_padding(self, data: bytes) -> bytes:
    if not data:
        return data

    padding_size = data[-1]
    if padding_size > 7:
        return data

    return data[:-padding_size]

def process_data(self, data: bytes, encrypt: bool = True) -> bytes:
    if encrypt:
        data = self._add_padding(data)

    processed_data = bytearray()

    # Processing in 8-byte blocks
    for i in range(0, len(data), 8):
        block_data = data[i:i+8]
        if len(block_data) < 8:
            block_data += bytes(8 - len(block_data))

        # Converting bytes into two 32-bit words
        left, right = struct.unpack('<2I', block_data)

        if encrypt:
            processed_left, processed_right =
self.encrypt_block((left, right))
        else:

```

```

        processed_left, processed_right =
self.decrypt_block((left, right))

        processed_data.extend(struct.pack('<2I', processed_left,
processed_right))

        if not encrypt:
            processed_data = self._remove_padding(bytes(processed_data))

        return bytes(processed_data)

    def encrypt_file(self, input_path: Union[str, Path], output_path:
Union[str, Path]):
        input_path = Path(input_path)
        output_path = Path(output_path)

        data = input_path.read_bytes()
        encrypted_data = self.process_data(data, encrypt=True)
        output_path.write_bytes(encrypted_data)

    def decrypt_file(self, input_path: Union[str, Path], output_path:
Union[str, Path]):
        input_path = Path(input_path)
        output_path = Path(output_path)

        data = input_path.read_bytes()
        decrypted_data = self.process_data(data, encrypt=False)
        output_path.write_bytes(decrypted_data)

    def validate_input(message: str, pattern: str, converter):
        while True:
            user_input = input(message).strip()
            if re.match(pattern, user_input):
                return converter(user_input)
            print("Invalid input. Please try again.")

    def main():
        print("GOST 28147-89 algorithm")

        cipher = GOST28147_89()

        while True:
            print("\nOptions:")
            print("1 - Encrypt file")
            print("2 - Decrypt file")
            print("3 - Exit")

            option = validate_input(
                "Choose option (1-3): ",
                r"^[1-3]$",
                int
            )

            if option == 3:
                break

            input_file = validate_input(
                "Enter input file path: ",
                r"^.+.$",
                str
            )

```

```

output_file = validate_input(
    "Enter output file path: ",
    r"^.+$$",
    str
)

try:
    if option == 1:
        cipher.encrypt_file(input_file, output_file)
        print(f"File successfully encrypted: {output_file}")
    elif option == 2:
        cipher.decrypt_file(input_file, output_file)
        print(f"File successfully decrypted: {output_file}")

except FileNotFoundError:
    print("Error: File not found")
except Exception as e:
    print(f"Error: {str(e)}")

if __name__ == "__main__":
    main()

```

ПРИЛОЖЕНИЕ Б

(обязательное)

Блок-схема алгоритма, реализующего программное средство

